

# A Formal Model for Dynamically Adaptable Services

Jorge Fox

**Abstract**—The growing complexity of software systems as well as changing conditions in their operating environment demand systems that are more flexible, adaptive and dependable. The service-oriented computing paradigm is in widespread use to support such adaptive systems, and, in many domains, adaptations may occur dynamically and in real time. In addition, services from heterogeneous, possibly unknown sources may be used. This motivates a need to ensure the correct behaviour of the adapted systems, and its continuing compliance to time bounds and other QoS properties. The complexity of dynamic adaptation (DA) is significant, but currently not well understood or formally specified. This paper elaborates a well-founded model and theory of DA, introducing formalisms written using COWS. The model is evaluated for reliability and responsiveness properties with the model checker CMC.

**Index Terms**—Service-Oriented Architectures, Dynamic Adaptation, Formal Methods.



## 1 INTRODUCTION

Modern software Systems typically operate in dynamic environments and are required to deal with changing operational conditions, while remaining compliant with the contracted quality of service. The execution context of modern distributed systems environments is not static but fluctuates dynamically, and to provide the expected functional service with the desired qualities, systems must be adaptable. Software service adaptation supports modification of existing services or inclusion of new ones, in response to inputs from the operating environment. Inputs or triggers for adaptation include changes in the running environment and availability of new services. When adaptation occurs at runtime, quality of service requirements range from timeliness, user perceived performance, service response time, to preservation of data integrity, service performance within given time bounds and the resulting system must comply with the execution time established for the system as a whole. Each one of these requirements has to be addressed accordingly, for instance, preservation of data integrity requires a mechanism to verify that the integrity of the data is kept during and after the adaptation, while timeliness and performance related requirements can be addressed by maintaining the execution times within the desired time bounds.

While the possibility of dynamic service deployment and evolution offers an exciting range of application development opportunities, it also poses a number of complex engineering challenges. These challenges include detecting adaptation triggers, facilitating timely,

dynamic service composition, predicting the temporal behaviour of unforeseen service assemblies and preventing adverse feature interactions following dynamic service composition. Therefore, a dynamically adaptable service has to be able to identify triggers for adaptation, select and conduct an adaptation strategy, and preserve desired quality of service properties, while avoiding undesired behaviours during or after the adaptation.

A promising approach to achieving the required properties in DA services is the use of formal methods and techniques, which have been successfully applied for managing complexity and system development to ensure implementations of high quality [1], verification and validation of the behaviour of adaptive programs [2], and architecture engineering processes that validate a program against desired functional properties [3]. However, most previous efforts to validate dynamically adaptable services against desired functional properties have been either limited to verifying conditions that are defined before runtime execution or impose high verification costs as each adaptation must be individually modelled and verified, making these approaches extremely expensive. Consider that an adaptive program with  $N$  different behaviours potentially has  $N^2$  distinct ways of adaptation. The result is that building DA services is either expensive or limited to predefined adaptation scenarios. Even more, these approaches focus their analysis on the resulting program neglecting the analysis of the adaptation mechanism itself, that is the adaptation manager (AM). We consider the adaptation mechanism to be of capital importance in the development of DA services, since it is responsible for monitoring compliance of the adaptation to desired properties and functionalities.

Current approaches to DA propose various mechanisms for handling adaptation, such as: Generic Interceptors [4], DA with aspect-orientation [5], Dynamic Reconfiguration [6], Dynamic Linking of Components

*This work was done during the tenure of an ERCIM "Alain Bensoussan" Fellowship Programme*

- Jorge Fox is with the Department of Telematics, Norwegian University of Science and Technology, Norway.  
E-mail: jfox@item.ntnu.no

[7], and Model-Driven Development of DA Software [2]. However, none of these provides a formal framework to verify the adaptation mechanism against commonly-accepted service properties. Formal languages provide the underpinnings to explain and model software systems in a precise manner, and are fundamental for the level of analysis, validation and proof required for assuring adaptation compliance to specifications. In this paper, we propose an abstract conceptual model of an AM and explore its operation in view of quality of service properties, such as availability and responsiveness.

This article is organised as follows: Section 2 provides an overview of current DA approaches. In order to build a formal model of the problem, an underlying language that facilitates its description and depicts the desired functionality properties at the same abstraction level is required, Section 3 discusses the steps we followed to select this foundation and the model checking tools supported by the language. Section 4 introduces our formal model and discusses its properties. Availability and responsiveness are identified as desirable attributes of services [8]. In Section 5 we explore our model in view of these attributes, and elaborate some lessons learned and topics requiring more research. Section 6 discusses related work. Finally, Section 7 presents some conclusions.

## 2 AN OVERVIEW OF DYNAMIC ADAPTATION TECHNIQUES

Some existing techniques for DA provide mechanisms to incorporate adaptation elements at design time, such as the work of [9] or allow for reconfigurations at runtime. These techniques can be classified according to the extent or scope of the adaptations that are made possible, the degree to which the adaptation triggers have to be known in advance, and the tools that the particular framework offers to implement or program a dynamically adaptive software. We also find techniques that allow dynamic linking and unlinking components or services as in [7] and techniques that apply aspect-orientation to achieve DA as [5]. Another group of techniques offer reconfiguration techniques that enable systems to adjust internal or global parameters to respond to changes in the environment as in [10], or Generic Interceptors with Adaptive CORBA, which enables message interception to add additional behaviour for adaptation [4].

While these techniques offer an interesting range of options to achieve different degrees of DA, questions related to adaptation trigger identification and soundness of a given adaptation model are still open.

## 3 FORMAL FOUNDATION

### 3.1 The Service-oriented Language COWS

In order to elaborate a model of dynamic adaptation, we explored a number of languages in [11]. Current service-

oriented formal languages like: PiDuce [12], SOCK-/JOLIE [13], COWS [14], KLAIM [15], and SCC [16] offer each a different range of possibilities to model DA. Every one of these languages has an underlying process algebra and constructs that support defining services and expressing substitution and deactivation processes. The prime mechanism in PiDuce to model service substitution is through virtual machines, while in the case of COWS it is modelled by delimited receiving and killing activities handled with its process calculus, JOLIE and PiDuce offer no deactivation process. After reviewing these languages, we concluded that the best-fit language for modelling our runtime dynamic adaptation problem is COWS. Given the characteristics of the languages selected, where only COWS provides constructs for timing analysis. Timeliness was not the only deciding criteria, considering only extent adaptation of services merely via channel renaming is sufficient to achieve DA is questionable, as is the case of PiDuce. In this regards the composition mechanisms of SOCK/JOLIE are more adequate, yet again with no possibility to evaluate timeliness. Our choice is clear and well founded. For more on the language COWS the reader may refer to [14].

### 3.2 Model Checking Tools

Model checking is an automatic technique for verifying finite-state reactive systems. Specifications are expressed in a propositional temporal logic, and the reactive system is modeled as a state-transition graph. An efficient search procedure is used to determine automatically if the specifications are satisfied by the state-transition graph [17].

Model checking [18] as an automatic verification technique covers a wide field of diverse, often ad hoc, and incomplete methods for showing correctness, or, more precisely, for finding bugs. Other verification techniques include theorem proving [19] and testing [20]. This technique allows software developers to find subtle errors in the design of safety-critical systems that often elude conventional simulation and testing techniques in a proven cost-effective manner by systematically exploring the state space of concurrent or reactive systems. Furthermore, model checking integrates well with conventional design methods. This is the reason why model checking is being adopted as a standard procedure for the quality assurance of reactive systems. This is done by computing a system's state space from an abstract description specified in a modeling language. Several properties of a model of a system can then be checked by exploring this state space: deadlocks, dead code, violations of user-specified assertions, etc. The properties that state-space exploration techniques can verify has been substantially broadened thanks to the development of model-checking methods for various temporal logics. A number of model checkers are available, such as: SPIN [21], SMV [22], UMC [23], and CMC [24], [25], as well as related analysis tools and languages such as the analyser Alloy [26].

CMC works on unmodified C or C++ implementations and explores large state spaces efficiently by storing states. Like traditional model checkers, CMC achieves the equivalent of executing astronomical numbers of tests in reasonable time. In this work, we use the model checker CMC [27] for the above mentioned reasons and its seamless integration with the language COWS.

## 4 A MODEL FOR DYNAMIC ADAPTATION

A wide range of currently available approaches for services adaptation offer several techniques to achieve DA. However, at the same time most are static, and more importantly, the flexibility of adaptation or the degree at which adaptations are achieved, is in most cases limited. Also, in many existing DA frameworks adaptation is achieved by parametrisation or reconfiguration, which may render limited solutions with respect to flexibility and limit further adaptations. Another area of opportunity we identify is the need to obtain service adaptation while ensuring compliance with predefined properties. This is precisely the motivation behind this work. We introduce in this work a model for DA that has been validated against desirable attributes of services and service-oriented computing applications [28]. According to this, a service should be: available, reliable, and responsive.

This section presents a two-step process for establishing our DA model. The five steps are listed as follows:

- 1) A scenario explaining a dynamic adaptation problem
- 2) Design of the model in a formal language suitable for verification

### 4.1 DA Scenario

The scenario is based on a tollbooth system, that can be considered a distributed system. The flow of events is given below. In this scenario a car approaches a tollbooth and gets a welcoming signal from it. If the car's payment protocol is not compatible to that of the tollbooth, then it receives a new protocol from the tollbooth. Afterwards, the car verifies the protocol and if needed adapts its electronic toll system to comply with the tollbooth's protocol. The scenario is illustrated in Figure 2.

The ETS has access to an electronic money pocket that is adjusted to a given tollbooth protocol, when the protocol is available. Modifying the system to comply with a new ETS protocol, steps 6-a and 6-b, in the tollbooth scenario, have to be performed at runtime. The adaptation is executed by a service that substitutes the former one. DA is the solution for this real-time adaptation problem.

### 4.2 Model of DA in COWS

The scenario in Figure 1 indicates the need for a mechanism to identify the adaptation trigger that initiates the adaptation process. We call it "Requestor", in our scenario it is represented by a welcoming signal and

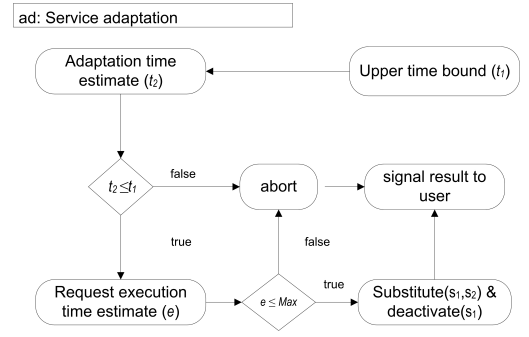


Fig. 2. Scenario. Service adaptation procedure

the request to confirm compatibility with the tollbooth. When the Requestor triggers an adaptation, this request has to be processed by another service, this service is the AM. The trigger for adaptation is processed and a decision on whether to proceed with an adaptation or not is taken also by the AM. Preservation of timeliness constraints has to be considered and incorporated in the AM, estimating the time to achieve adaptation and integrating this information with predefined upper-bound values. In case the adaptation can be realised within the specified time bounds, the AM may start a reconfiguration process.

The model illustrates the control flow for the adaptation, starting with the requestor, then a service `amcheck` is called with the time estimations. The timeliness conditions are examined in `amcheck` (listing 5), in case the adaptation can be fulfilled within the defined time bound, a second condition is evaluated (listing 6), which verifies that the adaptation preserves the overall execution time of the service. In case both conditions are uphold `amadapt` is called and the requestor receives the signal `signalOK` registering a successful adaptation (listing 4). Our model focused on two timeliness conditions, adaptation time and execution time, however, other conditions that the system has to preserve can be analysed by the AM in the same manner. For instance, verifying that the service to be reconfigured is in a quiescent state before performing any changes in order to avoid interaction and coordination conflicts.

## 5 VERIFYING QUALITY OF SERVICE PROPERTIES ON THE MODEL

Dynamic software architectures allow us to build dynamically adaptable systems by supporting the modification of a system's architecture at runtime. Possible modifications include structural adaptations that change the system configuration graph, for example, the creation of new and the deletion of existing components and connectors, as well as functional adaptations where components are replaced. Further, even more challenging changes are those that modify the behaviour of components and ultimately services.

In order to achieve reliable dynamic adaptable services we need to evaluate service modifications against a num-

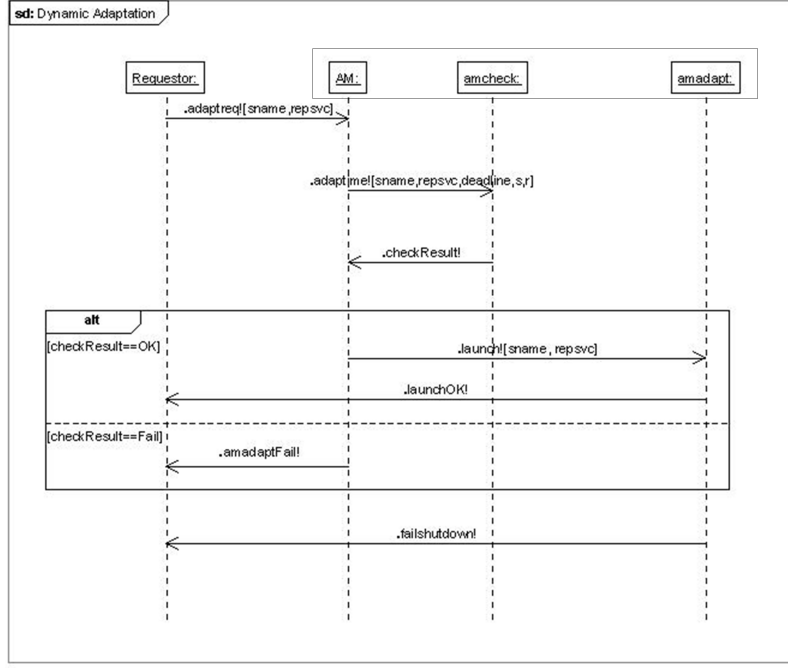


Fig. 1. Activity Diagram. Adaptation Process

ber of quality of service properties. Services must comply as a minimum with the following three properties. The first one, Responsiveness, means that the service always guarantees an answer to every received service request, unless the user cancels. The second property, Availability, requires that the service is always capable to accept a request. Finally, the third property, Reliability means that the service request can always succeed.

In order to evaluate a system against these properties, we first have to formulate them in a language capable of being analysed, preferably in an automated manner. In the following we introduce our representation of these properties in COWS-CMC. Afterwards, we explore the model at the hand of these properties with the model checker CMC evaluating its compliance to the properties and show a run of the model checker in Appendix B.

In order to explore the first property, Responsiveness, we define a formula with a universal quantifier on the response *amcheck* following a call to the AM specified as an existential quantifier to the AM, as shown in Listing 1. A run of the model checker can be found in Figure 3(a).

```

Responsiveness
AG[ request(adaptManager) ] true
EG[ accepting_request(adaptManager) ] AF[ response(amcheck) ]
true

```

Listing 1. Responsiveness

Availability can be easily represented by a universal quantifier on calls to the AM as illustrated in Listing 2.

```

Availability
AG[ request(adaptManager) ] true

```

Listing 2. Availability

A run of CMC to check Availability can be found in Figure 3(b).

```

Reliability
EG[ request(requestor) ] EG[ response(launchOK) ] EG[ response(
launchFail) ] true

```

Listing 3. Reliability

An examination of the model with the model-checker CMC confirms that it complies with the third property, Reliability, formulated in Listing 3. See Figure 3(a). A run of the model checker can be found in Figure 3(c).

## 6 RELATED WORK

We introduce related work in two groups. First, those approaches specifically related to DA, second, formal approaches that are applied to analyse similar families of problems as the ones presented here.

### Dynamic Adaptation

We find an overview of DA and its constituents in the work of Mckinley et al. [29], [30], however they do not advance a formal model or proposal to explore DA, which is the aims of our work. Similarly to the elements of DA we identified, Segarra and André describe a similar model to ours with components that can be customized for different applications, a component in their framework can be provided with a controller which performs the adaptation depending on execution conditions [31]. In our proposal we define one controller, the adaptation manager, that gathers information from supporting services such as timing and execution evaluation in order to perform adaptations. Work has also been carried out to map BPEL to Process Algebras as Ferrara

[32], to Pi-calculus as Abouzaid [33], and to Petri Nets as Ouyang et al. [34].

#### Formal approaches to DA

The work of Laneve and Zavattaro [35] on web services advances an extension to the  $\pi$ -calculus with a transaction construct, the calculus  $\text{web}\pi$ . This model supports time and asynchrony. However it remains at a more abstract level and is not applied to dynamic adaptation. Ferrara [32] relies on process algebra to design and verify web services, this work also allows to verify temporal logic properties as well as behavioural equivalence between services. Compared to this work, our attempt is more general and is directed at the study of dynamic adaptation. Finally our proposal is aimed at identifying a formal service-oriented language for modelling dynamic adaptation, rather than advancing techniques for formal verification of web services or services as in the work of Ferrara. Mori and Kita [36] explore the use of genetic algorithms to dynamic environments and offer a survey on problems of adaptation to dynamic environments. The work of ter Beek et al. [37] reviews service composition approaches with respect to a selection of service composition characteristics and helps to underscore the value of formal methods for service analysis at design, specially service composition. The authors present a valuable analysis of formal approaches to service composition and elaborate a useful comparison. We mentioned the need to provide mechanisms to assure consistency of the system during and after an adaptation, this has been further explored by Amano and Watanabe [38], at this stage, we do not aim at discussing consistency. Nevertheless by relying on a formal language we will be able to support consistency checks.

## 7 CONCLUSION

Real time DA is an area of research that poses new challenges to software development, considering software that may adapt to changing conditions in the operational environment, where new services may be added as they become available, or cope with reconfiguration issues, all this at runtime and under time constraints. DA has been proposed to provide solutions to these challenges. Proposing a methodology for the study of DA is still an open question. Formal methods have been in use for a long time in the computer science community and a number of new approaches and formal languages is available. Modelling DA with a formal language can provide precise answers to most of the existing questions and grant a better understanding of DA. In this work, we explored the use of the formal language COWS to model DA and try our model against three widely accepted service properties: Responsiveness, Availability and Reliability, with the model checker CMC, which integrates seamlessly with COWS.

## APPENDIX A

### ADAPTATION MANAGER MODEL IN COWS. LISTINGS

To provide some details of the adaptation model, we introduce here their listings.

```

let
  adaptManager(service) =
    * [X][Y][Z][XX][YY]
    service . create?<X,Y,Z,XX>.p . adapttime!<X,Y,Z,XX>
    | [X][Y][Z][XX]
    | ser . checkOK?<X,Y,Z,XX>.q . exectime!<Z,XX>
    | ser . checkFail?<>. ser . launchFail!<repsvc>
    | ser . checkFail2?<>. ser . launchFail!<repsvc>
  requestor() =
    serv . create!<0,4,10,60>
    | ser . checkOK2?<>.amadapt . launchOK!<> |
    (amadapt . launchOK?<>.s . signalOK!<>
    + ser . launchFail?<repsvc>.s . signalFail!<>
    + ser . launchFailx?<>.s . signalFail!<>)
in
  adaptManager(serv)
  | requestor()
  | * amcheck()
  | amcheck2()
  | s . signalFail?<>.nil
  | s . signalOK?<>.nil
end

```

Listing 4

Adaptation Manager, Requestor and Main Service

```

Amcheck_gt_deadline(X) =
  (ser . checkFail!<>)
Amcheck_le_deadline(X,Y,Z,XX) =
  (ser . checkOK!<X,Y,Z,XX>)
Amcheck_gt_deadline2(X) =
  (ser . checkFail2!<>)
  | memory . assert?<X>.nil
Amcheck_le_deadline2(X) =
  (ser . checkOK2!<>)
amcheck() =
  [X][Y][Z][XX]
  p . adapttime?<X,Y,Z,XX>.
  [i#]
  (i . selectgreater!<X gt Y> |
   (i . selectgreater?<true>.
    Amcheck_gt_deadline(X) +
    i . selectgreater?<false>.
    Amcheck_le_deadline(X,Y,Z,XX)
   )
  )

```

Listing 5

Adaptation-time check

```

amcheck2() =
  [X][Y]
  q . exectime?<X,Y>.
  [i#]
  [K]
  (i . selectgreater!<X gt Y> |
   (i . selectgreater?<true>.
    Amcheck_gt_deadline2(X) +
    i . selectgreater?<false>.
    Amcheck_le_deadline2(X)
   )
  )

```

Listing 6

Execution-time check

## APPENDIX B

### VERIFICATION OF PROPERTIES WITH CMC

The runs of the model checker CMC on the conditions specified in Section 5 follows.

```

C:\windows\system32\cmd.exe - cmc07q-Windows32-x86.exe
mc> EG (request(anchek)) I AG (response(selectgreater)) true
mc> -- Starting Evaluation with LTS Bound set to 8
The Formula: "EG (request(anchek)) I AG (response(selectgreater)) true"
is: TRUE
(states generated= 9, computations fragments generated= 17)
mc> why
-----
The formula:
EG ( request(anchek) ) I AG ( response(selectgreater) ) true
is FOUND_TRUE in State C1
because
C1 --> C2(create(0,4,10,60)) /* serv.create!<0, 4, 10, 60>.serv.create?<X, Y, Z, XX> */
C2 --> C3(adapttime(0,4,10,60)) /* p.adapttime!<0, 4, 10, 60>.p.adapttime?<X, Y, Z, XX> */
C3 --> C4(selectgreater(false)) /* i#1#.selectgreater!<false>,i#1#.selectgreater?<false> */
C4 --> C5(checkOK(0,4,10,60)) /* ser.checkOK!<0, 4, 10, 60>.ser.checkOK?<X, Y, Z, XX> */
C5 --> C6(exectime(10,60)) /* q.exectime!<10, 60>.q.exectime?<X, Y> */
C6 --> C7(selectgreater(false)) /* i#2#.selectgreater!<false>,i#2#.selectgreater?<false> */
C7 --> C8(checkOK2) /* ser.checkOK2!.ser.checkOK2? */
and the formula:
[ request(anchek) ] I AG [ response(selectgreater) ] true
is FOUND_TRUE in State C8
mc> =

```

(a) Reliability Check with CMC

```

C:\windows\system32\cmd.exe - cmc07q-Windows32-x86.exe
mc> AG(request(adaptManager)) true
mc> -- Starting Evaluation with LTS Bound set to 8
The Formula: "AG (request(adaptManager)) true"
is: TRUE
(states generated= 10, computations fragments generated= 37)
mc> AG(request(adaptManager)) true
mc> -- Starting Evaluation with LTS Bound set to 8
The Formula: "AG (request(adaptManager)) true"
is: TRUE
(states generated= 10, computations fragments generated= 37)
mc> AG(request(adaptManager)) true
mc> -- Starting Evaluation with LTS Bound set to 8
The Formula: "AG (request(adaptManager)) true"
is: TRUE
(states generated= 10, computations fragments generated= 37)
mc>

```

(b) Availability Check with CMC

```

C:\windows\system32\cmd.exe - cmc07q-Windows32-x86.exe
mc> EG(accepting_request(adaptManager)) I AF (response(anchek)) true
mc> -- Starting Evaluation with LTS Bound set to 8
The Formula: "EG (accepting_request(adaptManager)) I AF (response(anchek)) true"
is: TRUE
(states generated= 10, computations fragments generated= 54)
mc> why
-----
The formula:
EG ( accepting_request(adaptManager) ) I AF ( response(anchek) ) true
is FOUND_TRUE in State C1
because
C1 --> C2(create(0,4,10,60)) /* serv.create!<0, 4, 10, 60>.serv.create?<X, Y, Z, XX> */
C2 --> C3(adapttime(0,4,10,60)) /* p.adapttime!<0, 4, 10, 60>.p.adapttime?<X, Y, Z, XX> */
C3 --> C4(selectgreater(false)) /* i#1#.selectgreater!<false>,i#1#.selectgreater?<false> */
C4 --> C5(checkOK(0,4,10,60)) /* ser.checkOK!<0, 4, 10, 60>.ser.checkOK?<X, Y, Z, XX> */
C5 --> C6(exectime(10,60)) /* q.exectime!<10, 60>.q.exectime?<X, Y> */
C6 --> C7(selectgreater(false)) /* i#2#.selectgreater!<false>,i#2#.selectgreater?<false> */
C7 --> C8(checkOK2) /* ser.checkOK2!.ser.checkOK2? */
and the formula:
[ accepting_request(adaptManager) ] I AF [ response(anchek) ] true
is FOUND_TRUE in State C8
mc> =

```

(c) Responsiveness Check with CMC

Fig. 3. CMC Runs

## REFERENCES

- [1] M. Broy, J. Fox, F. Hölzl, D. Koss, M. Kuhrmann, M. Meisinger, B. Penzenstadler, S. Rittmann, B. Schätz, M. Spichkova, and D. Wild, *The Common Component Modeling Example: Comparing Software Component Models*. LNCS, 2009, no. 5153, ch. Service-Oriented Modeling of CoCoME with Focus and AutoFocus, to appear.
- [2] J. Zhang and B. H. C. Cheng, "Model-based development of dynamically adaptive software," in *ICSE '06: Proceeding of the 28th international conference on Software engineering*. New York, NY, USA: ACM, 2006, pp. 371–380.
- [3] M. H. ter Beek, A. Bucchiarone, and S. Gnesi, "Dynamic software architecture development: Towards an automated process," in *EUROMICRO-SEAA*. IEEE Computer Society, 2009, pp. 105–108.
- [4] S. M. Sadjadi and P. K. McKinley, "Act: An adaptive corba template to support unanticipated adaptation," *icdcs*, vol. 00, pp. 74–83, 2004.
- [5] Z. Yang, B. H. C. Cheng, R. E. K. Stirewalt, J. Sowell, S. M. Sadjadi, and P. K. McKinley, "An aspect-oriented approach to dynamic adaptation," in *WOSS '02: Proceedings of the first workshop on Self-healing systems*. New York, NY, USA: ACM, 2002, pp. 85–92.
- [6] M.-C. Pellegrini and M. Riveill, "Component management in a dynamic architecture," *J. Supercomput.*, vol. 24, no. 2, pp. 151–159, 2003.
- [7] C. Escoffier and R. S. Hall, "Dynamically adaptable applications with iPOJO service components," in *Software Composition*, ser. Lecture Notes in Computer Science, M. Lumpe and W. Vanderperren, Eds., vol. 4829. Springer, 2007, pp. 113–128.
- [8] A. Fantechi and R. P. F. T. Stefania Gnesi, Alessandro Lapadula Franco Mazzanti, "A model checking approach for verifying cows specifications," in *Proc. of Fundamental Approaches to Software Engineering (FASE'08)*, ser. Lecture Notes in Computer Science, vol. 4961. Springer, 2008, pp. 230–245.
- [9] R. Rouvoy, F. Eliassen, J. Floch, S. Hallsteinsen, and E. Stav, "Composing components and services using a planning-based adaptation middleware," in *Software Composition*, 2008, pp. 52–67.
- [10] J. M. Purtilo, "The polyolith software bus," *ACM Trans. Program. Lang. Syst.*, vol. 16, no. 1, p. 151174, 1994.
- [11] J. Fox and S. Clarke, "An analysis of formal languages for dynamic adaptation," in *International Conference on Engineering of Complex Computer Systems (ICECCS 2010)*, March 2010, pp. 3–13.
- [12] S. Carpineti, C. Laneve, and L. Padovani, "PiDuce – a project for experimenting web services technologies," *Science of Computer Programming*, vol. 74, no. 10, pp. 777–811, 2009.
- [13] C. Guidi, R. Lucchi, R. Gorrieri, N. Busi, and G. Zavattaro, "SOCK : A calculus for service oriented computing," *Proceedings of Service-Oriented Computing ICSOC 2006*, vol. 4294/2006, pp. 327–338, 2006.
- [14] A. Lapadula, R. Pugliese, and F. Tiezzi, "Cows: A timed service-oriented calculus," in *Proc. of 4th International Colloquium on Theoretical Aspects of Computing (ICTAC'07)*, ser. Lecture Notes in Computer Science, vol. 4711. Springer, 2007, pp. 275–290.
- [15] L. Bettini, V. Bono, R. D. Nicola, G. Ferrari, D. Gorla, M. Loreti, E. Moggi, R. Pugliese, E. Tuosto, and B. Venneri, "The klaim project: Theory and practice," in *GLOBAL COMPUTING: PROGRAMMING ENVIRONMENTS, LANGUAGES, SECURITY AND ANALYSIS OF SYSTEMS, VOLUME 2874 OF LNCS*. Springer-Verlag, 2003, pp. 88–150.
- [16] M. Boreale, R. Bruni, L. Caires, R. D. Nicola, I. Lanese, M. Loreti, F. Martins, U. Montanari, A. Ravara, D. Sangiorgi, V. T. Vasconcelos, and G. Zavattaro, "SCC: A service centered calculus," in *Web Services and Formal Methods*, ser. Lecture Notes in Computer Science, M. Bravetti, M. Núñez, and G. Zavattaro, Eds., vol. 4184. Springer, 2006, pp. 38–57.
- [17] E. M. Clarke, "Model cheking," in *FSTTCS*, ser. Lecture Notes in Computer Science, S. Ramesh and G. Sivakumar, Eds., vol. 1346. Springer, 1997, pp. 54–56.
- [18] E. M. Clarke, O. Grumberg, and D. A. Peled, *Model Checking*, M. Press, Ed., 2000.
- [19] F. Giunchiglia and P. Traverso, "Theorem proving in technology transfer: the user's point of view," *STTT*, vol. 3, no. 1, pp. 1–12, 2000.
- [20] A. Pretschner and M. Leucker, "Model-based testing-a glossary," in *Model-Based Testing of Reactive Systems*, ser. Lecture Notes in Computer Science, M. Broy, B. Jonsson, J.-P. Katoen, M. Leucker, and A. Pretschner, Eds., vol. 3472. Springer, 2004, pp. 607–609.
- [21] G. J. Holzmann, "Software model checking with spin," ser. *Advances in Computers*, M. Zelkowitz, Ed. Elsevier, 2005, vol. 65, pp. 77–108. [Online]. Available: <http://www.sciencedirect.com/science/article/B7RNF-4N69684-5/2/87a9e199a63acde757f3680bd5e95f5d>
- [22] *The SMV System*. [Online]. Available: <http://www.cs.cmu.edu/~modelcheck/smv.html>
- [23] J. Abreu, F. Mazzanti, J. L. Fiadeiro, and S. Gnesi, "A model-checking approach for service component architectures," in *FMOODS/FORTE*, ser. Lecture Notes in Computer Science, D. Lee, A. Lopes, and A. Poetzsch-Heffter, Eds., vol. 5522. Springer, 2009, pp. 219–224.
- [24] M. Musuvathi, D. Y. W. Park, A. Chou, D. R. Engler, and D. L. Dill, "Cmc: a pragmatic approach to model checking real code," in *OSDI '02: Proceedings of the 5th symposium on Operating systems design and implementation Copyright restrictions prevent ACM from being able to make the PDFs for this conference available for downloading*. New York, NY, USA: ACM, 2002, pp. 75–88.
- [25] M. Musuvathi, A. Chou, D. L. Dill, and D. Engler, "Model checking system software with cmc," in *EW 10: Proceedings of the 10th workshop on ACM SIGOPS European workshop*. New York, NY, USA: ACM, 2002, pp. 219–222.
- [26] D. Jackson, "Alloy: a lightweight object modelling notation," *ACM Trans. Softw. Eng. Methodol.*, vol. 11, no. 2, pp. 256–290, 2002. <http://fmt.isti.cnr.it/cmc/>.
- [27] G. Alonso, F. Casati, H. A. Kuno, and V. Machiraju, *Web Services - Concepts, Architectures and Applications*, ser. Data-Centric Systems and Applications. Springer, 2004.
- [28] P. K. McKinley, S. M. Sadjadi, E. P. Kasten, and B. H. Cheng, "Composing adaptive software," *Computer*, vol. 37, no. 7, pp. 56–64, 2004.
- [29] P. K. McKinley, S. M. Sadjadi, E. P. Kasten, and B. H. C. Cheng, "A taxonomy of compositional adaptation," Dept. Computer Science and Engineering, Michigan State University, Tech. Rep. MSU-CSE-04-17, 2004.
- [30] M.-T. Segarra and F. André, "A framework for dynamic adaptation in wireless environments," in *TOOLS '00: Proceedings of the Technology of Object-Oriented Languages and Systems (TOOLS 33)*. Washington, DC, USA: IEEE Computer Society, 2000, p. 336.
- [31] A. Ferrara, "Web services: a process algebra approach," in *ICSOC '04: Proceedings of the 2nd International Conference on Service Oriented Computing*. New York, NY, USA: ACM, 2004, pp. 242–251.
- [32] F. Abouzaid, "A mapping from pi-calculus into bpel," in *Proceeding of the 2006 conference on Leading the Web in Concurrent Engineering*. Amsterdam, The Netherlands, The Netherlands: IOS Press, 2006, pp. 235–242.
- [33] C. Ouyang, E. Verbeek, W. M. P. van der Aalst, S. Breutel, M. Dumas, and A. H. M. ter Hofstede, "Formal semantics and analysis of control flow in ws-bpel," *Sci. Comput. Program.*, vol. 67, no. 2-3, pp. 162–198, 2007.
- [34] C. Laneve and G. Zavattaro, "Foundations of web transactions." Springer, 2005, pp. 282–298.
- [35] K. H. Mori Naoki, "Genetic algorithms for adaptation to dynamic environments - a survey," in *26th Annual Conference of the IEEE Electronics Society IECON 2000*, I. P. I. E. Conference), Ed., 2000.
- [36] A. B. Maurice H. ter Beek and S. Gnesi, "Formal methods for service composition," *Annals of Mathematics, Computing & Teleinformatics*, vol. 1, 5, pp. 1–10, 2007.
- [37] W. T. Amano Noriki, "Towards constructing component-based software systems with safe dynamic adaptability," in *International Workshop on Principles of Software Evolution (IWPSE)*, 2001.